Neural network approach to scaling analysis of critical phenomena

Ryosuke Yoneda [*] and Kenji Harada [†]
Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan

(Received 7 September 2022; accepted 28 March 2023; published 27 April 2023)

Determining the universality class of a system exhibiting critical phenomena is one of the central problems in physics. There are several methods to determine this universality class from data. As methods to collapse plots onto scaling functions, polynomial regression, which is less accurate, and Gaussian process regression, which provides high accuracy and flexibility but is computationally expensive, have been proposed. In this paper, we propose a regression method using a neural network. The computational complexity is linear only in the number of data points. We demonstrate the proposed method for the finite-size scaling analysis of critical phenomena in the two-dimensional Ising model and bond percolation problem to confirm the performance. This method efficiently obtains the critical values with accuracy in both cases.

DOI: 10.1103/PhysRevE.107.044128

I. INTRODUCTION

Critical phenomena have been a significant research topic in statistical mechanics for many years. The scaling behavior near a critical point plays a central role in critical phenomena because critical phenomena of different physical systems share it. We can identify the universality class of critical phenomena by the scaling exponents called critical exponents. The universality class depends on only the dimensionality, the symmetry, and the type of interaction between components, not the details of systems. The renormalization group supports this property of the universality class for equilibrium systems [1–6]. Also, in the case of nonequilibrium, the universality of critical phenomena has been extensively studied [7]. Therefore, identifying the universality class of critical phenomena is theoretically and experimentally important.

Even in the case of theoretical models, it is difficult to derive the value of critical exponents analytically in general. Thus, it is important to numerically estimate the value of critical exponents from the given data. In particular, we often use the finite-size scaling (FSS) law to determine the value of critical exponents from data for finite-size systems [8]. We have applied the FSS analysis to various critical phenomena from classical and quantum systems [9–14]. The physical quantity near a critical point in a finite-size system obeys the scaling law written as

$$A(T, L) = L^{-c_2} F[(T - T_c)L^{c_1}], \tag{1}$$

where $A(T, L)$ is a physical quantity at temperature T in a finite-size system whose size is L. T_c is the critical temperature. The exponents c_1 and c_2 are critical exponents. Here $F[\cdot]$ is a scaling function. Unfortunately, we do not know the scaling function's form in advance. Thus, if we directly use the FSS law, we need to infer not only the value of the critical temperature and exponents but also the scaling function itself from the given data. In order to avoid this problem, several methods have been proposed. For example, we use the Binder ratio to determine the effective critical temperature for each system size and so on. However, in the case of hard problems, the amount of data is often limited in a narrow region near a critical point. Thus, using all data, not only data in the narrow region, is useful for estimating critical values. Here, we consider the direct use of the FSS law by collapsing data onto a scaling function. The approach is called FSS in the following.

The classical way to perform FSS has been to assume that the scaling function is polynomial and then use the least-squares method to determine the critical exponents by fitting the scaling law to data. However, this method has the problem of determining the degree of the polynomial while preventing overfitting. It also requires high accuracy in a very narrow region near the critical point because the representation power of polynomials is poor. To deal with this problem, a Bayesian regression method [15,16] was proposed to infer the scaling function and critical exponents, in which the scaling function is a sample of the adjusted Gaussian process. In this method, we assume only the smoothness of the scaling function. Thus, we can use data to fit in the broader range near a critical point. This type of regression method is called Gaussian process (GP) regression in the machine learning field and is widely used to analyze real data [17]. The GP for FSS [15], called Bayesian scaling analysis (BSA) in the following, has been widely used to determine critical exponents and scaling functions for various critical phenomena because of its high accuracy [12–14,18–26]. However, the computational cost of the GP is a problem because the computation of a likelihood gradient needs the inverse matrix of the covariance of the GP. The size of the matrix is $N \times N$, where N is the number of data points. The total cost of the GP is proportional to N^3. Thus, it is expensive when the size of the data increases.

This paper proposes a neural network approach to scaling analysis (NSA), a FSS method using a neural network. The

[*] yoneda.ryosuke.o95@kyoto-u.jp
[†] harada.kenji.8e@kyoto-u.ac.jp

main idea is to model the scaling function with a neural network and train it with data. Because of the greater expressive power of a neural network, it can represent scaling functions more appropriately than polynomials [27]. This method's advantage is that it is computationally less expensive than the GP because the computational complexity is linear to the number of data points. With the success of machine learning [28], neural networks are being used in many research areas to solve various problems in physics [29]. Although some studies use machine learning to detect the phase transition [30–32], this paper uses neural networks for FSS to determine critical points and exponents.

We organize this paper as follows. Section II briefly reviews BSA and neural networks. Section III introduces the NSA and some techniques for stabilizing the learning process. Section IV demonstrates the NSA for the two-dimensional Ising model and the two-dimensional bond percolation. Section V introduces a method to calculate the confidence interval of critical values with the NSA. Finally, Sec. VI gives the conclusions and a discussion.

II. PRELIMINARIES

A. Bayesian scaling analysis

Using the rescaled variables

$$X \equiv (T - T_c)L^{c_1}, \quad Y \equiv A(T, L)L^{c_2}, \tag{2}$$

the FSS law is rewritten as follows:

$$Y = F[X]. \tag{3}$$

Thus, the FSS analysis is an inference of critical parameters, $\boldsymbol{\theta}_c = (T_c, c_1, c_2)$, so that rescaled data points, $(\boldsymbol{X}_{\boldsymbol{\theta}_c}, \boldsymbol{Y}_{\boldsymbol{\theta}_c})_i = (X_i, Y_i)$, collapse on a scaling function F.

The basic idea of the BSA is to assume that the scaling function is a sample of the GP and apply GP regression [17]. The scaling function is sampled from the GP as

$$F \sim \mathcal{GP}\big(0, k_{\boldsymbol{\theta}_h}\big). \tag{4}$$

This means

$$F[\boldsymbol{X}] = (F[X_1], \ldots, F[X_N]) \sim \mathcal{N}(\boldsymbol{0}, \Sigma), \tag{5}$$

where $\mathcal{N}(\boldsymbol{0}, \Sigma)$ is a multivariate normal distribution with a zero mean vector and the covariance matrix Σ is

$$(\Sigma)_{ij} = k_{\boldsymbol{\theta}_h}(X_i, X_j), \quad i = 1, \ldots, N, \tag{6}$$

where $k_{\boldsymbol{\theta}_h}$ is a kernel function and $\boldsymbol{\theta}_h$ is a hyperparameter vector used to define it. Therefore, the BSA regards the rescaled data points in the FSS law as an N-dimensional vector of a multivariate normal distribution,

$$\boldsymbol{Y}_{\boldsymbol{\theta}_c} \sim \mathcal{N}(\boldsymbol{0}, \Sigma_{\boldsymbol{\theta}}), \tag{7}$$

where $\boldsymbol{\theta} = (\boldsymbol{\theta}_c, \boldsymbol{\theta}_h)$. Then the conditional probability of data points given parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_h, \boldsymbol{\theta}_c)$ is

$$p(\boldsymbol{Y}_{\boldsymbol{\theta}_c} \mid \boldsymbol{\theta}) = \frac{1}{\sqrt{\det(2\pi\,\Sigma_{\boldsymbol{\theta}})}} \exp\left[-\frac{1}{2}\boldsymbol{Y}_{\boldsymbol{\theta}_c}^{\top}\Sigma_{\boldsymbol{\theta}}^{-1}\boldsymbol{Y}_{\boldsymbol{\theta}_c}\right]. \tag{8}$$

Assuming that the prior distribution of parameters $\boldsymbol{\theta}$ is uniform for simplicity, we have

$$p(\boldsymbol{\theta} \mid \boldsymbol{Y}_{\boldsymbol{\theta}_c}) \propto p(\boldsymbol{Y}_{\boldsymbol{\theta}_c} \mid \boldsymbol{\theta}) \tag{9}$$

from Bayes's theorem. Therefore, the most probable parameters $\boldsymbol{\theta}$ are the maximum of the log-likelihood function,

$$\mathcal{L}(\boldsymbol{\theta}) = \ln\left[p(\boldsymbol{\theta} \mid \boldsymbol{Y}_{\boldsymbol{\theta}_c})\right] \tag{10}$$

$$= -\tfrac{1}{2}\ln\left[\det(2\pi\,\Sigma_{\boldsymbol{\theta}})\right] - \tfrac{1}{2}\boldsymbol{Y}_{\boldsymbol{\theta}_c}^{\top}\Sigma_{\boldsymbol{\theta}}^{-1}\boldsymbol{Y}_{\boldsymbol{\theta}_c}. \tag{11}$$

One way to find the maximum of the log likelihood is the gradient method. The gradient of the log-likelihood function reads

$$\frac{\partial\mathcal{L}}{\partial\theta} = -\frac{1}{2}\mathrm{tr}\left(\Sigma_{\boldsymbol{\theta}}^{-1}\frac{\partial\Sigma_{\boldsymbol{\theta}}}{\partial\theta}\right) - \boldsymbol{Y}_{\boldsymbol{\theta}_c}^{\top}\Sigma_{\boldsymbol{\theta}}^{-1}\frac{\partial\boldsymbol{Y}_{\boldsymbol{\theta}_c}}{\partial\theta} + \frac{1}{2}\big(\Sigma_{\boldsymbol{\theta}}^{-1}\boldsymbol{Y}_{\boldsymbol{\theta}_c}\big)^{\top}\frac{\partial\Sigma_{\boldsymbol{\theta}}}{\partial\theta}\big(\Sigma_{\boldsymbol{\theta}}^{-1}\boldsymbol{Y}_{\boldsymbol{\theta}_c}\big). \tag{12}$$

Gradient descent methods, such as Adam [33], allow us to obtain the desired critical parameters successfully. If we take the kernel in the GP as the radial basis function kernel,

$$k_{\boldsymbol{\theta}_h}(X_i, X_j) \equiv \theta_{h,1}\exp\left[-\frac{|X_i - X_j|^2}{2(\theta_{h,2})^2}\right], \tag{13}$$

sample functions are infinitely differentiable [34]. Therefore, the BSA assumes only the smoothness of the scaling function and hence applies to many systems. However, the gradient (12) includes the inverse calculation of the covariance matrix $\Sigma_{\boldsymbol{\theta}}$. The inverse matrix calculation is generally numerically unstable and has a computational cost of $O(N^3)$. For this reason, the BSA has the disadvantage of being computationally expensive when the size of the data increases.

B. Neural networks

We briefly introduce neural networks and their application to regression problems, which have played a crucial role in recent machine learning achievements [28].

In this paper, we consider a fully connected neural network with layers numbered from 0 to M, each containing $n_0, \ldots, n_M$ neurons. The network function $F_{\mathsf{NN}} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_M}$ can be written with the following recurrence relation:

$$\alpha^0(x) \equiv x, \tag{14}$$

$$\tilde{\alpha}^{l+1}(x) \equiv W^l\alpha^l(x) + b^l, \tag{15}$$

$$\alpha^{l+1}(x) \equiv \phi(\tilde{\alpha}^{l+1}(x)), \tag{16}$$

$$F_{\mathsf{NN}}(x) \equiv \tilde{\alpha}^M(x). \tag{17}$$

Here, α^l and $\tilde{\alpha}^l$ are functions from $\mathbb{R}^{n_0}$ to $\mathbb{R}^{n_l}$ for $l = 0, 1, \ldots, M$. $W^l \in \mathbb{R}^{n_{l+1}\times n_l}$ and $b^l \in \mathbb{R}^{n_{l+1}}$ are weight matrices and bias vectors for $l = 0, 1, \ldots, M-1$. Each element is a trainable parameter and initialized with $W^l_{i,j} \sim \mathcal{N}(0, n_l^{-1})$ and $b^l_i \sim \mathcal{N}(0, 1)$. The total number of training parameters is $\sum_{l=0}^{M-1} n_{l+1}(n_l + 1)$. $\phi : \mathbb{R} \to \mathbb{R}$ is a nonlinear activation function applied entrywise for a multidimensional input.

Now let us see how we do a regression analysis with neural networks. We have a training data set $\mathcal{D} = \{(X_i, Y_i)\}_{i=1}^N$ with $X_i \in \mathbb{R}^{n_0}$ and $Y_i \in \mathbb{R}^{n_M}$, and we are going to find a function f that satisfies

$$Y_i \sim \mathcal{N}\big[f(X_i), E_i^2\big], \tag{18}$$

with some errors E_i for $i = 1, 2, \ldots, N$. We assume that the function $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_M}$ can be well approximated by a neural network function $F_{\mathsf{NN}} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_M}$ because of the universal approximation property of neural networks [35,36]. Then, the Gaussian negative log-likelihood loss function is

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\left[\frac{\{Y_i - F_{\mathsf{NN}}(X_i)\}^2}{E_i^2} + \ln\left(2\pi E_i^2\right)\right]. \tag{19}$$

We find the desired function by minimizing the loss function by varying parameters in the neural network F_{NN}. Unlike linear regression, we do not have an analytical result for the parameters, but using the stochastic gradient descent method [37] allows us to obtain minimized parameters numerically. The complicated gradient calculation for neural network parameters is done by automatic differentiation [38], which was implemented in recent deep learning frameworks, including JAX [39] and PyTorch [40]. Also, the computational cost of the loss function (19) is $O(N)$ with respect to the number of data N, which is less expensive than the Gaussian process regression. A comparison of the Gaussian process regression and the neural network approach is made in Sec. A 1.

III. NEURAL SCALING ANALYSIS

Let us now introduce the NSA. We explain how the NSA is conducted to obtain critical exponents from data. We also give tips on stabilizing the learning process.

A. Data preparation

We first prepare the data set $\mathcal{D} = \{(T_i, L_i, A_i, \delta A_i)\}_{i=1}^{N}$, where A_i is the ith observable data point for temperature T_i and system size L_i with statistical error δA_i.

We next perform the data preprocessing. In machine learning, data normalization is essential for efficient learning. We rescale the data as

$$L \mapsto \frac{1}{L_{\max}}L, \tag{20}$$

$$T \mapsto \frac{2}{T_{\max} - T_{\min}}T - \frac{T_{\max} + T_{\min}}{T_{\max} - T_{\min}}, \tag{21}$$

$$A \mapsto \frac{1}{A_{\max} - A_{\min}}A, \tag{22}$$

$$\delta A \mapsto \frac{1}{A_{\max} - A_{\min}}\delta A, \tag{23}$$

where $L_{\max}$ is the largest L. $T_{\max}$ and $T_{\min}$ and $A_{\max}$ and $A_{\min}$ are the largest and smallest values of T and A for system size $L_{\max}$, respectively. Then, as in Eqs. (21) and (22), the temperature T is mapped in the interval $[-1, 1]$ for $L_{\max}$, and the observable A is mapped so that the width is 1 for $L_{\max}$. δA is transformed in the same way as A in Eq. (23). This transformation means that for the scaling function F at the largest system size $L_{\max}$ in Eq. (1), the data width of the input $(T - T_c)L^{c_1}$ is shaped to be 2, and the data width of the output $A(T, L)L^{c_2}$ is shaped to be 1.

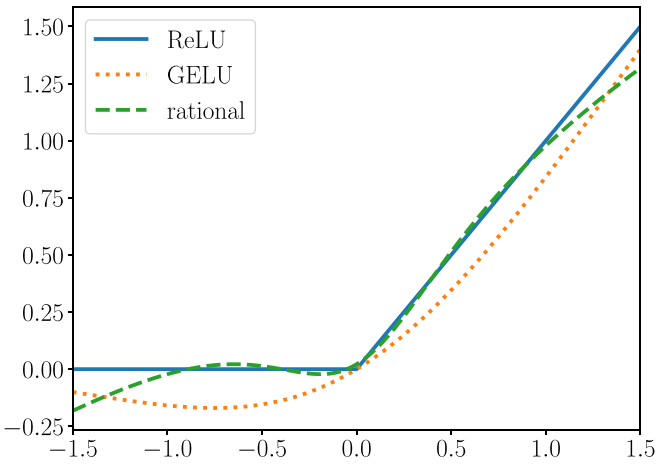

FIG. 1. Comparison of three activation functions: the ReLU function, GELU function [41], and rational activation function with $(r_P, r_Q) = (3, 2)$ [42]. For the rational activation function, the parameters a_i and b_j in Eq. (25) are initialized so that the L^∞ distance of the ReLU function and the rational activation function on $[-1, 1]$ is minimized.

B. Scaling function

The basic idea is to replace the scaling function with a neural network. Since the scaling function is $F : \mathbb{R} \to \mathbb{R}$, we set $n_0 = n_M = 1$ for the neural network function F_{NN}. The number of layers M and the layer size n_i for $i = 1, \ldots, M - 1$ are left as hyperparameters, and we set $M = 3$ and $n_1 = n_2 = 20$ in the experiment.

We remark on the choice of the nonlinear activation function. In many neural network use cases, the Rectified Linear Unit (ReLU) function, $\phi_{\mathrm{ReLU}}(x) = \max\{x, 0\}$, is employed as the activation function to avoid gradient explosion or disappearance. However, the ReLU function is not differentiable at the origin, which conflicts with our requirement that the scaling function is smooth. To solve this problem, we use the Gaussian Error Linear Unit (GELU) function [41],

$$\phi_{\mathrm{GELU}}(x) = x\Phi(x), \tag{24}$$

which is a smoothed version of the ReLU function. Here, Φ is the cumulative distribution function of the standard normal distribution. We also consider the rational activation function [42], which has the form

$$\phi(x) = \frac{P(x)}{Q(x)} = \frac{\sum_{i=0}^{r_P} a_i x^i}{\sum_{j=0}^{r_Q} b_j x^j}, \tag{25}$$

where $P(x)$ and $Q(x)$ are polynomials with degrees r_P and r_Q. The coefficients of the polynomials a_i and b_j are trainable parameters. We use $(r_P, r_Q) = (3, 2)$ as recommended in the original paper [42]. See Fig. 1 for a comparison of the above activation functions.

C. Training the scaling function

We set up a loss function to learn the scaling function, critical temperature, and critical exponents [43]. Given Eq. (1) for a neural network with Eq. (2), we require the scaling function to satisfy the following equation for the data:

$$Y_i \sim \mathcal{N}\left[F_{\mathsf{NN}}(X_i), E_i^2\right], \tag{26}$$

where X_i, Y_i, and E_i are

$$X_i = (T_i - T_c)L_i^{c_1}, \tag{27}$$

$$Y_i = L_i^{c_2}A_i, \quad E_i = L_i^{c_2}\delta A_i \tag{28}$$

for $i = 1, 2, \ldots, N$. We remark that the statistical error E_i can vary for X_i since physical systems that exhibit critical phenomena tend to show large data fluctuation near the critical point. The Gaussian negative log-likelihood loss function, Eq. (19), treats such data with input-dependent noises [43]. In practice, the variance E_i^2 is replaced with $\max(\epsilon, E_i^2)$ with a threshold $\epsilon = 10^{-6}$ for computational stability, as implemented in PyTorch. We note that for homoscedastic data where $E_i = E$ for all i, the Gaussian negative log-likelihood loss function becomes the least-squares regression loss function.

The goal of the learning is to find the parameter that gives the minimum value for the loss function $\mathcal{L}$. We use stochastic gradient descent methods, particularly Adam [33], to update parameters and search for optimal solutions. We summarize tips obtained through actual experiments. The learning rate set by default in Adam's training, $\alpha = 10^{-3}$, is the one recommended for training neural networks. Therefore, in learning the loss function $\mathcal{L}$ in Eq. (19), the learning rate for the parameters of the neural network and the learning rate for the critical exponents c_1 and c_2 and critical points T_c should be different so that the learning can converge in fewer iterations. In experiments, we set the learning rate for c_1, c_2, and T_c as $\alpha_c = 10^{-2}$.

We also note how to handle the data. A typical setup for machine learning is batch learning, where the neural network weights are learned using all the data to compute the gradient. However, this method has a problem: if there is a large amount of data, the time required for each learning step becomes enormous. We usually use minibatch learning to solve this problem, which reduces the time needed for gradient calculation by taking a subset of the whole data set when calculating the gradient. This method is also well known to make learning less stagnant and optimization more successful. However, we confirmed that batch learning is sufficient for learning in the NSA. In all of the following experiments, we use batch learning.

D. Clipping parameters

In many cases, the signs of the critical exponents c_1 and c_2 are known in advance when performing the FSS. Also, with the above rescaling of the temperature T, we know that T_c lies between -1 and 1. Therefore, we need a way to optimize parameters within an interval or half-open interval. Here, unconstrained optimization is achieved by constructing a bijective map from the real number $\mathbb{R}$ to the interval. First, for optimization on the half-open interval (a, ∞), we use the following bijective map:

$$y = g_{(a,\infty)}(x) \equiv \phi_{\mathrm{SP}}(x) + a, \tag{29}$$

where $\phi_{\mathrm{SP}}(x) = \ln(1 + e^x)$ is another smooth approximation of the ReLU function. Second, for the half-open interval $(-\infty, a)$, we use the following bijective map:

$$y = g_{(-\infty,a)}(x) \equiv -\phi_{\mathrm{SP}}(x) + a. \tag{30}$$

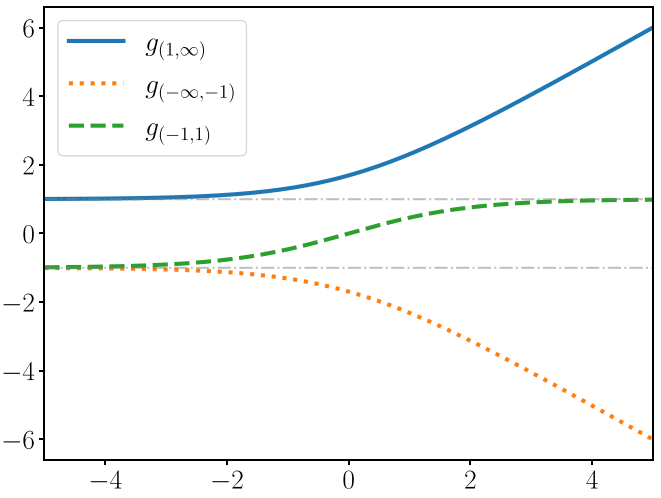

FIG. 2. Comparison of three bijective maps: $g_{(1,\infty)}$, $g_{(-\infty,-1)}$, and $g_{(-1,1)}$. The gray dotted lines are the 1 and -1 guiding lines.

Finally, the bijective map that achieves parameter optimization on the interval (a, b) is as follows:

$$y = g_{(a,b)}(x) \equiv a + (b - a)\sigma(x), \tag{31}$$

where $\sigma(x) = 1/(1 + e^{-x})$ is a sigmoid function.

All of these functions $g_{(a,\infty)}$, $g_{(-\infty,a)}$, and $g_{(a,b)}$ are bijective maps (see Fig. 2 for a comparison), and their inverse maps are analytically calculated as follows:

$$x = g^{-1}_{(a,\infty)}(y) \equiv \ln(e^{y-a} - 1), \tag{32}$$

$$x = g^{-1}_{(-\infty,a)}(y) \equiv \ln(e^{a-y} - 1), \tag{33}$$

$$x = g^{-1}_{(a,b)}(y) \equiv \ln(y - a) - \ln(b - y). \tag{34}$$

IV. NUMERICAL EXPERIMENTS

This section demonstrates the NSA for the two-dimensional Ising model and two-dimensional bond percolation [44]. In both cases, we can compare the results of the NSA with the exact results of those models. We discuss the performance comparison between the NSA and BSA in the Appendix.

A. Two-dimensional Ising model

We first consider the two-dimensional Ising model on a square lattice, where the Hamiltonian and its probability distribution are

$$\mathcal{H}(\boldsymbol{s}) = -J \sum_{\langle i,j \rangle} s_i s_j, \tag{35}$$

$$P(\boldsymbol{s}) = \frac{1}{Z} \exp\left[-\mathcal{H}(\boldsymbol{s})/(k_B T)\right]. \tag{36}$$

Here, s_i denotes the spin variable of the ith site with $s_i = \pm 1$, $\langle i, j \rangle$ denotes the nearest neighbor pairs, and J denotes the positive coupling constant. Z is the partition function, and k_B is the Boltzmann constant. In the following, we set $J/k_B = 1$ for simplicity. The order parameter is a magnetization defined as $M = \sum_i s_i$. The two-dimensional Ising model on a square lattice is known to be solvable [45]; the critical point

is given by $\frac{1}{T_c} = \frac{1}{2}\ln(1+\sqrt{2}) = 0.44068\ldots$, and its critical exponents are $\beta = 1/8$, $\gamma = 7/4$, and $\nu = 1$.

In this paper, we perform FSS using the *Binder ratio* to accurately determine the phase transition point. The Binder ratio is defined as the kurtosis of the order parameter, which reads

$$U = \frac{1}{2}\left(3 - \frac{\langle M^4\rangle}{\langle M^2\rangle^2}\right), \tag{37}$$

where $\langle\cdot\rangle$ denotes the average over the canonical distribution (36). Since the Binder ratio is a dimensionless quantity and does not require scaling, its FSS form has only one critical exponent ν, which reads

$$U(T,L) = \Psi_B[(1/T - 1/T_c)L^{1/\nu}]. \tag{38}$$

We first do the Monte Carlo simulations to obtain the Binder ratio. See [15] for the detailed settings. Now we have the data set $\mathcal{D} = \{(1/T_i, L_i, U_i, \delta U_i)\}_i$, where U_i is the ith Binder ratio with the inverse temperature $1/T_i$ and system size L_i. δU_i is the statistical error of the ith Binder ratio U_i. The data are shown in Fig. 3(a) for system size $L = 64$, 128, and 256. Then to perform the NSA, we rescale the data as in Sec. III A. The scaling function is set to a neural network with layer sizes $n_0 = n_3 = 1$ and $n_1 = n_2 = 20$, and the loss function is set to the Gaussian negative log-likelihood function (19) with

$$X_i = (1/T_i - 1/T_c)L_i^{c_1}, \tag{39}$$

$$Y_i = U_i, \quad E_i = \delta U_i, \tag{40}$$

where the critical values are clipped with $c_1 = g_{(0,\infty)}(\theta_1)$ and $1/T_c = g_{(-1,1)}(\theta_c)$.

We optimize the scaling function and θ_1 and θ_c with the optimizer Adam [33] for 10^4 iteration times. Figure 3(b) shows the result of the NSA. We see that all the scaled data are in the scaling function obtained by a neural network. Figure 4 shows the loss $\mathcal{L}$ and the critical values through 10^4 iterations. We see that the loss values are well converged through iterations. The results of the critical values are $c_1 = 0.99078$ and $1/T_c = 0.44070$, which are very close to the exact results.

B. Two-dimensional bond percolation

Next, we demonstrate the NSA for the two-dimensional bond percolation model. In percolation theory, bonds between lattice points open with probability p; then two points are connected. We discuss the clusters that are then formed, their size, and their dependence on the probability [46]. It is known from current extensive research that as the probability of the opening of bonds between lattice points increases, a phase transition phenomenon is observed in which clusters have formed and spread throughout the entire system with a certain probability p_c. The value of this critical probability p_c is known for many two-dimensional lattices. See Fig. 5 for typical configurations of two-dimensional bond percolation on a square lattice with different values of p [46].

In this paper, we performed FSS using the *truncated mean cluster size* $\chi^f(p)$, which is defined by the mean size of finite open clusters. By denoting the cluster containing the origin by C and writing its size as $|C|$, we have

$$\chi^f(p) = \mathbb{E}_p[|C|; |C| < \infty]. \tag{41}$$

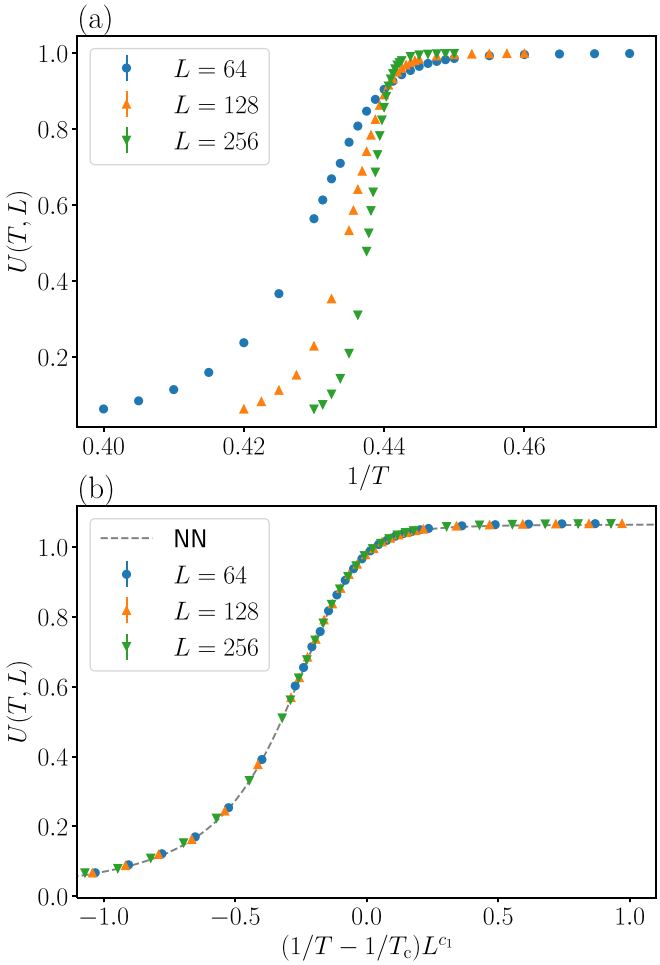

FIG. 3. (a) The Binder ratios of the Ising model on a square lattice for system size $L = 64$, 128, and 256. Error bars are so small that they may not be visible. (b) Result of FSS using NSA for the Binder ratios shown in (a). The gray dashed curve shows the scaling function obtained with a neural network. We note that the data are scaled as in Sec. III A.

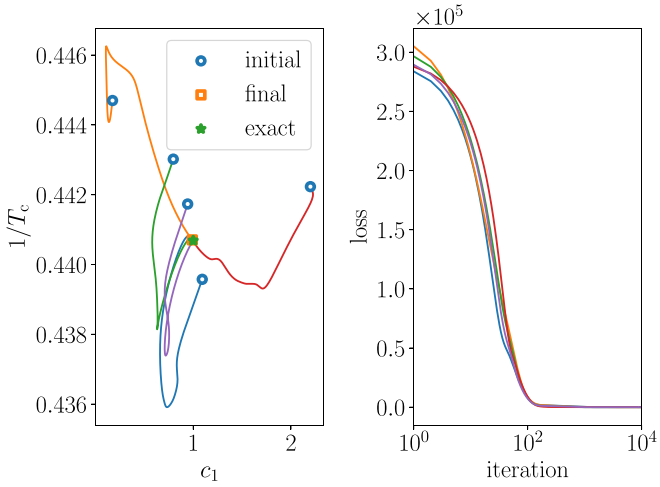

FIG. 4. Critical values $(c_1, 1/T_c)$ (left) and the loss value (right) from the learning process of the Binder ratio of the two-dimensional Ising model with five different initial values.

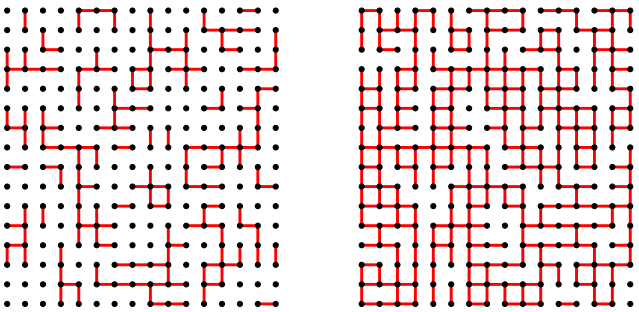

FIG. 5. Typical bond percolation configurations on a square lattice with bond probabilities $p = 0.3$ (left) and $p = 0.6$ (right) for lattice size $L = 16$. At $p = 0.3$, no clusters are spread throughout the system. When $p = 0.6$ exceeds the critical probability $p_c = 1/2$, clusters spreading throughout the system, i.e., clusters crossing up and down or left and right, can be confirmed.

Near the critical probability p_c, the truncated mean cluster size is believed to behave as $\chi^f(p) \approx |p - p_c|^{-\gamma}$ as $p \to p_c$ with a critical exponent γ. Therefore, its FSS form reads

$$\chi^f(p, L) = L^{\gamma/\nu}\Phi[(p - p_c)L^{1/\nu}] \tag{42}$$

with a scaling function Φ. For two-dimensional bond percolation on a square lattice, the critical probability is $p_c = 1/2$, and the critical exponents are $\gamma = 43/18$ and $\nu = 4/3$ [46].

We note the difference between the mean cluster size $\chi(p)$ and the truncated mean cluster size $\chi^f(p)$. The mean cluster size is defined as $\chi(p) = \mathbb{E}_p[|C|]$. In the subcritical phase $p < p_c$, it is expected that $\chi(p) \approx (p_c - p)^{-\gamma}$ for $p \uparrow p_c$. However, in the supercritical phase $p > p_c$, $\chi(p) = \infty$ from the definition of the critical probability p_c, which makes $\chi(p)$ difficult to handle theoretically. For the truncated mean cluster size, it is expected that $\chi^f(p) \approx (p - p_c)^{-\gamma'}$ for $p \downarrow p_c$, where we assume that $\gamma = \gamma'$. Since $\chi(p) = \chi^f(p)$ in the subcritical phase, the formula $\chi^f(p) \approx |p - p_c|^{-\gamma}$ allows us to treat both phases in a unified manner.

To perform FSS, we first do the Monte Carlo simulations to obtain the truncated mean cluster sizes. Cluster sizes can be calculated efficiently using a *disjoint-set data structure* and the *union-find algorithm*. We now have the data set $\mathcal{D} = \{(p_i, L_i, \chi_i^f, \delta\chi_i^f)\}_i$, where χ_i^f is the ith truncated mean cluster size with probability p_i and system size L_i. $\delta\chi_i^f$ is the statistical error of χ_i^f. The data are shown in Fig. 6(a) for system size $L = 64$, 128, and 256 with 1000 trials. The total number of data points is 150. The rescaling of the data set is done as in Sec. III A, and the neural network setting is the same as the one with the two-dimensional Ising model in Sec. IV A. The loss function is set to the Gaussian negative log-likelihood function (19) with

$$X_i = (p_i - p_c)L_i^{c_1}, \tag{43}$$

$$Y_i = L_i^{c_2}\chi_i^f, \quad E_i = L_i^{c_2}\delta\chi_i^f, \tag{44}$$

where the critical values are clipped with $c_1 = g_{(0,\infty)}(\theta_1)$, $c_2 = g_{(-\infty,0)}(\theta_2)$, and $p_c = g_{(-1,1)}(\theta_c)$. The exact values of the two-dimensional bond percolation problem are $p_c = 1/2 = 0.5$, $c_1 = 3/4 = 0.75$, and $c_2 = -43/24 = -1.7916\ldots$. By optimizing the loss function with the Adam optimizer,

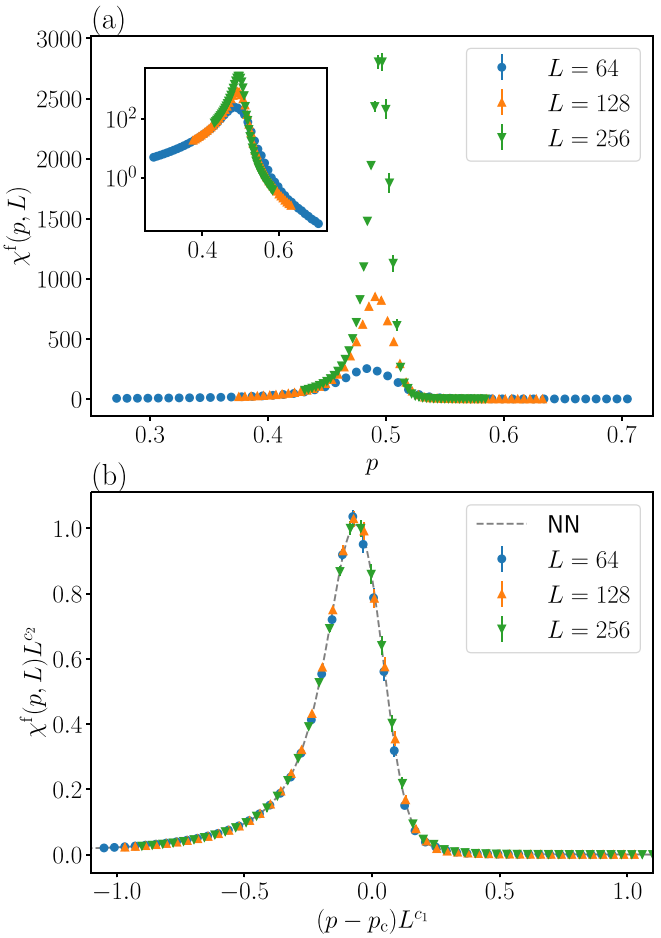

FIG. 6. (a) The truncated mean cluster sizes $\chi^f(p, L)$ of the bond percolation on a two-dimensional square lattice for system size $L = 64$, 128, and 256. Each point is the result of 1000 trials. Error bars are so small that they may not be visible. The inset shows the log scale of $\chi^f(p, L)$. (b) Result of FSS using NSA for the truncated mean cluster sizes shown in (a). The gray dashed curve shows the scaling function obtained with a neural network. We note that the data are scaled as in Sec. III A.

we have successfully obtained the critical probability $p_c = 0.5002$ and the critical exponents $c_1 = 0.7540$ and $c_2 = -1.763$. Figure 7 shows the loss $\mathcal{L}$ and the critical values through 10^4 iterations. In Fig. 6(b), we see that the scaled data are collapsed on a single curve, and the scaling function is well approximated by the trained neural network.

V. ACCURACY OF THE ESTIMATION OF CRITICAL VALUES

In the previous section, we confirmed that the critical values could be estimated using a neural network based on numerical data of critical phenomena. However, this is only one estimation result, and we do not know how much we can trust this result. Most studies that numerically determine the critical value provide confidence intervals for its value. How should we calculate the confidence interval by the NSA proposed in this paper?

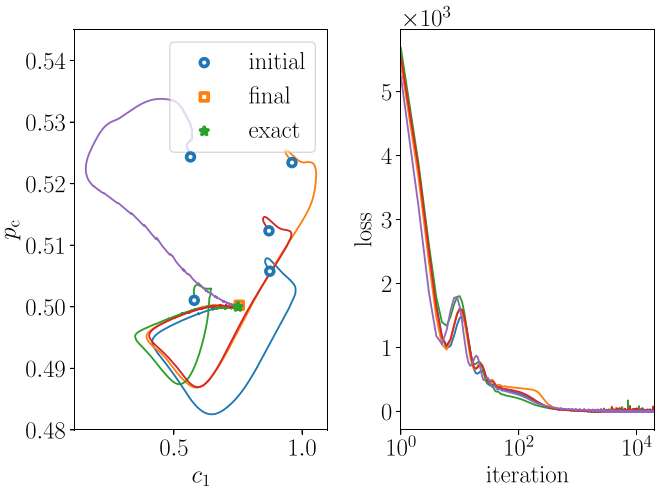

FIG. 7. Critical values (c_1, p_c) (left) and the loss value (right) through the learning process of the truncated mean cluster size of the two-dimensional bond percolation model with five differential initial values.

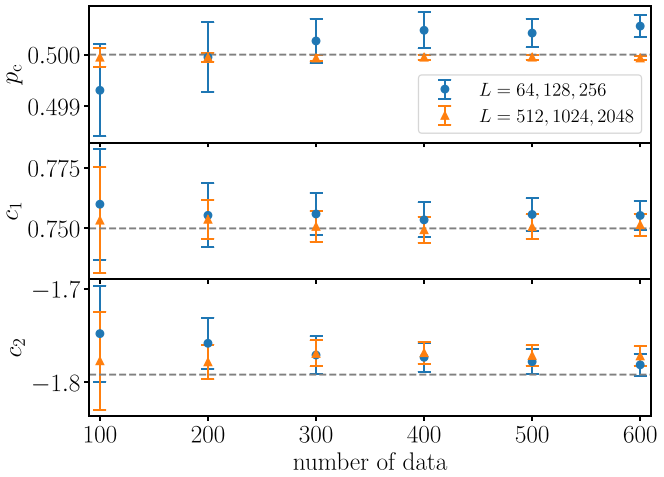

FIG. 8. Learned critical values c_1 and c_2 and the critical point p_c for the two-dimensional bond percolation model with confidence intervals indicated by error bars. We vary the total number of data from 100 to 600. Blue circles represent the system size triplet $L = 64, 128, 256$, and orange triangles represent the system size triplet $L = 512, 1024, 2048$. Confidence intervals are calculated by the bootstrap approach with 500 resamplings. True values are plotted with gray dashed lines.

Here, we consider using a bootstrap-like approach to calculate confidence intervals for critical values [47]. The bootstrap approach is based on the following procedure: First, we randomly resample the data of the critical phenomena with replacement [48]. Next, we take random initial values for the parameters learned in the NSA. The weights of the neural network are initialized randomly as described in Sec. II B, and the parameters of the critical values c_1, c_2, and T_c are chosen uniformly at random from the appropriate interval. Using the data set and the initial values of the parameters prepared in this way, we obtain the critical values using the NSA. We perform these procedures several times and calculate the standard deviation for a series of critical values, setting this as a 1σ confidence interval for the critical values.

For example, let us determine the confidence interval of the critical values of the two-dimensional bond percolation model. We use the truncated mean cluster size χ^{f} used in Sec. IV B for our data. For the initial values of the critical values p_c, c_1, and c_2, we choose uniformly at random from intervals [0.432, 0.583], [0.6, 0.9], and $[-1.9, -1.6]$, respectively. Under this configuration, we calculate the critical values 500 times and obtain the confidence interval. In Fig. 8, we use the truncated mean cluster size for the data with system size triplets $L = 64, 128, 256$ and $L = 512, 1024, 2048$. We vary the total data size from 100 to 600 and plot the learned critical values with 1σ confidence intervals. As can be seen in Fig. 8, the error bar becomes smaller as the data size increases, indicating that we get a more accurate result for a larger number of data points for the two-dimensional percolation model. We also observe that for p_c and c_1, the accuracy with respect to the true value improves as the system size L of the data used for training increases.

VI. CONCLUSIONS AND DISCUSSION

We proposed a method for the scaling analysis of critical phenomena using neural networks. The basic idea of this method is to model the scaling function by a neural network. For example, we consider FSS analysis, but we can apply it to general scaling analysis, which contains an unknown scaling function. The method is computationally less expensive than the previously proposed method for scaling analysis using GP regression, making it a simpler method.

We demonstrated it for the two-dimensional Ising and bond percolation models. We accurately obtained the critical points and exponents, and the data were well collapsed to the learned scaling function for both cases. Because the computational complexity is linear for the number of data points, we can handle the FSS analysis efficiently.

Using a bootstrap-like approach, we also checked the robustness of the estimation results of our method and defined the confidence intervals as the standard deviation of estimated values. They are consistent with the exact values. Another approach to determine the accuracy of the critical values would be to use the *stochastic gradient Langevin dynamics* [49], which achieves Bayesian learning by adding noise to the gradient method. It has the advantage of being a Bayesian approach but scales linearly in computational complexity for the number of data.

ACKNOWLEDGMENTS

The authors thank the anonymous referees for their valuable comments and for making the manuscript better. R.Y. acknowledges the support of JSPS KAKENHI Grant No. JP22J15552. K.H. acknowledges the support of JSPS KAKENHI Grant No. JP20K03766 and a Grant-in-Aid for Transformative Research Areas "The Natural Laws of Extreme Universe – A New Paradigm for Spacetime and

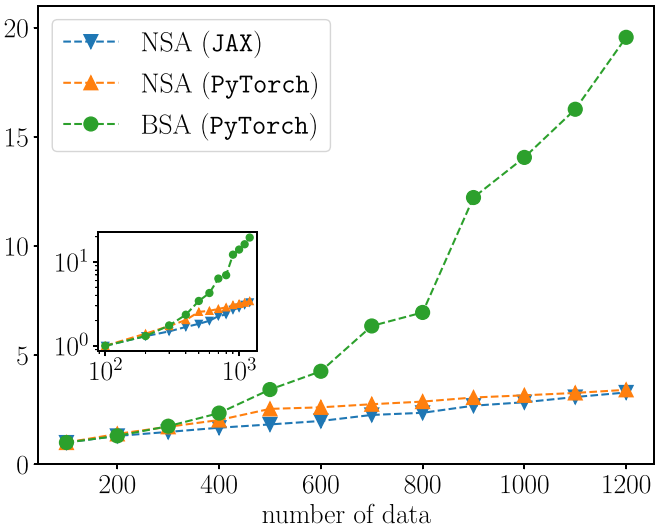

FIG. 9. Relative average execution time for one epoch where we scale the execution time for 100 data points as 1. The data used are the truncated mean cluster size of the two-dimensional bond percolation. The inset shows the log-log plot.

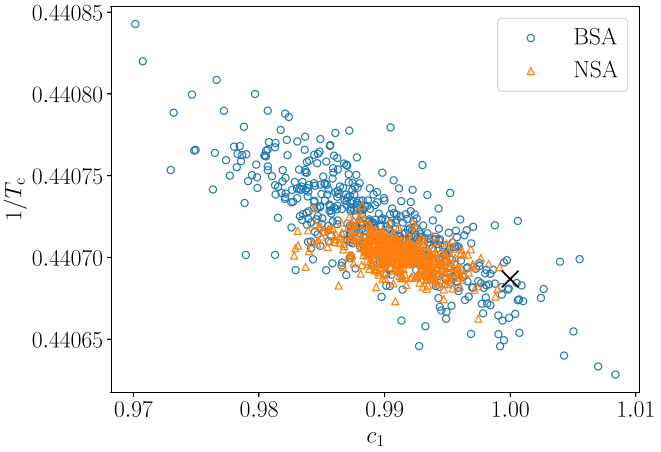

FIG. 10. Final learned value distributions of critical values $(c_1, 1/T_c)$ for NSA and BSA for the two-dimensional Ising model. The cross represents the theoretical value $c_1 = 1$ and $1/T_c = \frac{1}{2}\ln(1+\sqrt{2})$.

Matter from Quantum Information" (KAKENHI Grants No. JP21H05182 and No. JP21H05191) from JSPS of Japan.

APPENDIX: COMPARISON BETWEEN NSA AND BSA

In this Appendix, we compare the performance of the NSA and BSA, particularly with respect to computation time and the accuracy of critical values.

1. Computation time

Let us compare the execution time of the NSA with BSA with respect to the number of data points. PyTorch and JAX implementations were used to execute the NSA. PyTorch implementation was used to execute BSA. The execution environment is as follows: macOS 12.5.1, Apple M1 Max, 32 GB memory. In Fig. 9, we measure the (average) computation time per epoch for the NSA and BSA and plot how much the computation time increases as the number of data points increases, taking the case with 100 data points as 1. As can be seen, the computation time of the NSA is linear to the number of data points, while BSA shows a larger growth rate of computation time with respect to the number of data points. This is because BSA requires $O(N^3)$ computational complexity due to the inverse matrix calculation, while the NSA requires only linear $O(N)$ computational complexity. We note that the JAX implementation uses JAX.JIT(), which performs *just in time* compilation [50,51], so it allows for faster execution than PyTorch.

2. Accuracy of critical values

We next calculate the confidence intervals of critical values for the two-dimensional Ising model and compare the accuracy of the NSA and BSA methods. For data used in Sec. IV A, we resample the data with replacement and calculate the critical values 500 times. Figure 10 shows the results of bootstrap for NSA and BSA. We observe that the learned results are more concentrated in the NSA than the BSA. For the BSA method, $c_1 = 0.989(5)$, and $1/T_c = 0.44071(3)$, whereas for the NSA method, $c_1 = 0.991(2)$ and $1/T_c = 0.44070(1)$.

[1] K. G. Wilson, Renormalization group and critical phenomena. I. Renormalization group and the Kadanoff scaling picture, Phys. Rev. B **4**, 3174 (1971).
[2] K. G. Wilson, Renormalization group and critical phenomena. II. Phase-space cell analysis of critical behavior, Phys. Rev. B **4**, 3184 (1971).
[3] K. G. Wilson and J. Kogut, The renormalization group and the ϵ expansion, Phys. Rep. **12**, 75 (1974).
[4] K. G. Wilson, The renormalization group and critical phenomena, Rev. Mod. Phys. **55**, 583 (1983).
[5] N. Goldenfeld, *Lectures on Phase Transitions and the Renormalization Group* (CRC Press, Boca Raton, FL, 2018).
[6] J. Cardy, *Scaling and Renormalization in Statistical Physics*, Cambridge Lecture Notes in Physics, Vol. 5 (Cambridge University Press, Cambridge, 1996).
[7] M. Henkel, H. Hinrichsen, S. Lübeck, and M. Pleimling, *Non-equilibrium Phase Transitions* (Springer, Dordrecht, 2008), Vol. 1.
[8] J. Cardy, *Finite-Size Scaling* (Elsevier, Amsterdam, 1988).
[9] K. Binder, Finite size scaling analysis of Ising model block distribution functions, Z. Phys. B **43**, 119 (1981).
[10] K. Slevin and T. Ohtsuki, Corrections to Scaling at the Anderson Transition, Phys. Rev. Lett. **82**, 382 (1999).
[11] L. Wang, K. S. D. Beach, and A. W. Sandvik, High-precision finite-size scaling analysis of the quantum-critical point of

$S = 1/2$ Heisenberg antiferromagnetic bilayers, Phys. Rev. B **73**, 014431 (2006).

[12] K. Harada, T. Suzuki, T. Okubo, H. Matsuo, J. Lou, H. Watanabe, S. Todo, and N. Kawashima, Possibility of deconfined criticality in SU(N) Heisenberg models at small N, Phys. Rev. B **88**, 220408(R) (2013).

[13] Y. Otsuka, S. Yunoki, and S. Sorella, Universal Quantum Criticality in the Metal-Insulator Transition of Two-Dimensional Interacting Dirac Electrons, Phys. Rev. X **6**, 011029 (2016).

[14] R. Yoneda, K. Harada, and Y. Y. Yamaguchi, Critical exponents in coupled phase-oscillator models on small-world networks, Phys. Rev. E **102**, 062212 (2020).

[15] K. Harada, Bayesian inference in the scaling analysis of critical phenomena, Phys. Rev. E **84**, 056704 (2011).

[16] K. Harada, Kernel method for corrections to scaling, Phys. Rev. E **92**, 012106 (2015).

[17] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning* (MIT Press, Cambridge, MA, 2006), Vol. 2.

[18] J. Nasu and Y. Motome, Thermodynamics of Chiral Spin Liquids with Abelian and Non-Abelian Anyons, Phys. Rev. Lett. **115**, 087203 (2015).

[19] R. Singh, J. H. Bardarson, and F. Pollmann, Signatures of the many-body localization transition in the dynamics of entanglement and bipartite fluctuations, New J. Phys. **18**, 023046 (2016).

[20] S. Hesselmann and S. Wessel, Thermal Ising transitions in the vicinity of two-dimensional quantum critical points, Phys. Rev. B **93**, 155157 (2016).

[21] J. D'Emidio and R. K. Kaul, New Easy-Plane $\mathbb{CP}^{N-1}$ Fixed Points, Phys. Rev. Lett. **118**, 187202 (2017).

[22] T. Horita, H. Suwa, and S. Todo, Upper and lower critical decay exponents of Ising ferromagnets with long-range interaction, Phys. Rev. E **95**, 012143 (2017).

[23] F. Ferrari, S. Bieri, and F. Becca, Competition between spin liquids and valence-bond order in the frustrated spin-1/2 Heisenberg model on the honeycomb lattice, Phys. Rev. B **96**, 104401 (2017).

[24] S. Iino, S. Morita, N. Kawashima, and A. W. Sandvik, Detecting signals of weakly first-order phase transitions in two-dimensional Potts models, J. Phys. Soc. Jpn. **88**, 034006 (2019).

[25] T. C. Lang and A. M. Läuchli, Quantum Monte Carlo Simulation of the Chiral Heisenberg Gross-Neveu-Yukawa Phase Transition with a Single Dirac Cone, Phys. Rev. Lett. **123**, 137602 (2019).

[26] Y. Nomura and M. Imada, Dirac-Type Nodal Spin Liquid Revealed by Refined Quantum Many-Body Solver Using Neural-Network Wave Function, Correlation Ratio, and Level Spectroscopy, Phys. Rev. X **11**, 031034 (2021).

[27] The representation power of a neural network is used to model various quantities in physics and chemistry, for example, the quantum amplitude of ground states of quantum many-body systems [52] and the static local field correction of the warm, dense electron gas [53].

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, Cambridge, MA, 2016).

[29] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, and L. Zdeborová, Machine learning and the physical sciences, Rev. Mod. Phys. **91**, 045002 (2019).

[30] L. Wang, Discovering phase transitions with unsupervised learning, Phys. Rev. B **94**, 195105 (2016).

[31] T. Ohtsuki and T. Ohtsuki, Deep learning the quantum phase transitions in random two-dimensional electron systems, J. Phys. Soc. Jpn. **85**, 123706 (2016).

[32] J. Carrasquilla and R. G. Melko, Machine learning phases of matter, Nat. Phys. **13**, 431 (2017).

[33] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, in *Proceedings of the 3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA*, Vol. 1 (2015).

[34] M. Kanagawa, P. Hennig, D. Sejdinovic, and B. K. Sriperumbudur, Gaussian processes and kernel methods: A review on connections and equivalences, arXiv:1807.02582.

[35] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. **2**, 303 (1989).

[36] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Networks **4**, 251 (1991).

[37] S. Ruder, An overview of gradient descent optimization algorithms, arXiv:1609.04747.

[38] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, Automatic differentiation in machine learning: A survey, J. Mach. Learn. Res. **18**, 1 (2018).

[39] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, JAX: Composable transformations of Python+NumPy programs, http://github.com/google/jax.

[40] A. Paszke *et al.*, PyTorch: An imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Curran Associates, 2019), pp. 8024–8035.

[41] D. Hendrycks and K. Gimpel, Gaussian error linear units (GELUs), arXiv:1606.08415.

[42] N. Boullé, Y. Nakatsukasa, and A. Townsend, Rational neural networks, in *Advances in Neural Information Processing Systems*, Vol. 33 (2020), pp. 14243–14253.

[43] D. Nix and A. Weigend, Estimating the mean and variance of the target probability distribution, in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)* (IEEE, Piscataway, NJ, 1994).

[44] The code implemented in PYTHON is available on GitHub together with the data set. JAX implementation, https://github.com/yonesuke/jaxfss; PyTorch implementation, https://github.com/KenjiHarada/FSS-tools.

[45] R. Baxter, *Exactly Solved Models in Statistical Mechanics* (Dover, Mineola, NY, 2007).

[46] G. Grimmett, *Percolation* (Springer, Berlin, 1999).

[47] B. Efron, Bootstrap methods: Another look at the jackknife, Annal. Stat. **7**, 1 (1979).

[48] We assume that the physical quantities and their confidence intervals are already given, and we randomly select the same number of elements with replacement. In the case of two-dimensional bond percolation, we first compute the truncated mean cluster sizes for the given system sizes and probabilities with 1000 trials to obtain their confidence intervals. We

denote these data as $\mathcal{D} = \{(p_i, L_i, \chi_i^{\mathrm{f}}, \delta\chi_i^{\mathrm{f}})\}_{i=1}^{n_{\mathrm{data}}}$. From the data $\mathcal{D}$, we randomly select the same number of elements with replacement.

[49] M. Welling and Y. W. Teh, Bayesian learning via stochastic gradient Langevin dynamics, in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)* (Association for Computing Machinery, New York, 2011), pp. 681–688.

[50] Just in time compilation with JAX, https://jax.readthedocs.io/en/latest/jax-101/02-jitting.html.

[51] XLA: Optimizing Compiler for Machine Learning, https://www.tensorflow.org/xla.

[52] G. Carleo and M. Troyer, Solving the quantum many-body problem with artificial neural networks, Science **355**, 602 (2017).

[53] T. Dornheim, J. Vorberger, S. Groth, N. Hoffmann, Z. A. Moldabekov, and M. Bonitz, The static local field correction of the warm dense electron gas: An ab initio path integral Monte Carlo study and machine learning representation, J. Chem. Phys. **151**, 194104 (2019).